TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 1802R022 – Informatika a logistika

Oprava překlepů zadávaných do vyhledávače (Seznam.cz)

Spelling correction for search engine queries (Seznam.cz)

Bakalářská práce

Autor: **Jan Holub**
Vedoucí práce: Ing. Igor Kopetschke
Konzultant: Štěpán Škrob

V Liberci 9. 5. 2008

Originál zadání práce

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Tímto bych chtěl poděkovat všem, kteří přispěli ke vzniku této bakalářské práce. Především pak panu Štěpánu Škrobovi a Mgr. Radimu Řehůřkovi z firmy Seznam.cz za odborné rady a poskytnutí všech potřebných materiálů.

Abstrakt

Internetové vyhledávače jsou hlavním nástrojem, který umožňuje rychlé a efektivní vyhledávání informací na internetu. Aby však poskytly hledané a relevantní údaje, je vždy potřeba správně formulovat dotaz. Jedním z častých důvodů nesprávně zadaného dotazu jsou slova s překlepy nebo různými pravopisnými chybami. V tomto případě není vyhledávač schopen poskytnout požadované informace a namísto toho většinou vrátí jen neužitečné a nepoužitelné výsledky. Jedním z možných řešení, jak uživateli usnadnit hledání, je integrovat do vyhledávače algoritmus pro opravu překlepů, který nabídne možné opravy pro nesprávně zadaný dotaz. Tato práce se zabývá návrhem takového algoritmu a jeho možným využitím v českém internetovém vyhledávači Seznam.cz.

Hlavním zdrojem dat pro správnou opravu překlepů jsou záznamy uživatelských dotazů zadávaných do vyhledávače, které poskytují mnoho důležitých poznatků a informací. Nejdůležitější z nich je četnost, s jakou se jednotlivá slova v dotazech vyskytují. Prvním krokem bylo tedy zpracovat tyto logy a získat z nich požadovanou statistiku unigramů a bigramů, která byla využita při samotné opravě. Druhý samostatný zdroj dat pak tvoří ověřený slovník.

Navrhovaný algoritmus lze rozdělit na dvě hlavní části. V té první se snaží nalézt všechny možné kandidáty na opravu vstupního dotazu. Druhá část pak slouží k výběru nejvhodnějšího kandidáta, který je ve výsledku nabídnut uživateli jako možná oprava jeho dotazu. To vše se musí realizovat v co nejkratším možném čas, který je v tomto případě velmi důležitý.

Vlastní implementace algoritmu je provedena jako konzolová aplikace, která se spustitelná na systému Windows a napsaná v jazyce C. Využívá mimo jiné ternární vyhledávací strom a Damerau-Levenshteinovu vzdálenost.

Abstract

Search engines are a main tool for a fast and efficient information retrieval on the web. In order to provide correct and relevant results, an input query must be well formed. But many queries are wrong because of misspelled words. In this case search engine returns inconclusive and useless informations. One of the possible solutions how to simplify searching for users is integrates spelling correction into the search engine. This work presents proposal for algorithm that implements spelling correction for the Czech search engine Seznam.cz.

Main source of data exploited for spelling correction are logs of user queries that provide a large amount of information about language. The most important of them is a frequency of the words. First step consists in parsing all logs and creating statistics of unigrams and bigrams that forms foundation for the algorithm. Second source of information is a trusted language dictionary.

Proposed algorithm can be divided into the two main parts. First part tries to find all possible candidates for a correction of the input query string. In the second part, algorithm selects the best candidate that is offered to the user as a possible reform of his query. The whole operation has to be performed as fast as possible, because time is crucial.

The algorithm for spelling correction is implemented as a console application for an operating system Windows and written in the C language. Among other things, it's based on a ternary search trees data structure and Damerau-Levenshtein distance.

Obsah

1. Úvod

1.1. Internetové vyhledávače

Milióny lidí využívají internet pro získávání potřebných informací a právě internetové vyhledávače představují hlavní nástroj, který jim v tom pomáhá. Vyhledávače musí denně zpracovat obrovské množství dotazů (Seznam.cz přes 10 miliónů dotazů za den), což představuje obrovský prostor pro možné chyby a překlepy. Nějakou chybu související s nesprávně zapsaným výrazem obsahuje téměř každý desátý dotaz. Pro takové dotazy vyhledávač obvykle vrátí jen malé množství odpovídajících stránek, které navíc obsahují stejnou chybu, jakou měl hledaný výraz. Ve výsledku ovšem většinou chybí známé a spolehlivé weby, které zpravidla podobné překlepy neobsahují. Zabudovaná oprava překlepů, která uživatele okamžitě informuje o možných chybách v zadaném dotazu a nabídne opravu, tak může přinést značné posílení jeho snahy o nalezení požadovaných informací.

První vyhledávače nabízející okamžitou opravu překlepů se začaly objevovat v dubnu roku 2001 a patřily mezi ně dnes velmi oblíbený Google, AltaVista nebo také Excite [5]. Podrobnější technické informace o použitých algoritmech nejsou ovšem známy, jelikož to daným společnostem poskytuje významné výhody nad ostatními vyhledávači. Vše ale napovídá tomu, že používají běžné metody s využitím statistických informací o četnosti slov, získaných ze záznamu již provedených dotazů.

1.2. Tradiční přístup pro opravu překlepů

Oprava překlepů má dlouholetou historii a tradičně se zaměřuje na řešení typografických chyb, jako jsou *vložení*, *smazání*, *nahrazení* a *přehození* písmen, které mají za následek neznámé slovo, tj. takové slovo, které se nenachází ve slovníku daného jazyka. Tradiční programy na opravu překlepů hledají pro každé neznámé slovo malý seznam slovníkových alternativ, které jsou následně uživateli nabídnuty jako možná oprava. Při tom se spoléhají na informace o frekvenci, s jakou se daná slova používají, na obvyklé klávesové chyby (psaní *m* místo *n* apod.) a na fonetické chyby na úrovni slov i písmen [3]. Velmi málo spell checkerů se pokouší detekovat a opravit chyby při záměně slov, které jsou sice ve správném slovníkovém tvaru, ale v nesprávném kontextu. Pro vyřešení tohoto problému bylo již vynaloženo

určité úsilí, jehož výsledkem jsou například systémy, které spoléhají na syntaktické zákonitosti pro zjištění substitučních chyb.

Dřívější přístupy byly také založené na nereálném předpokladu, že všechna možná použití slov jsou známá. To představuje určité problémy, protože mnoho substitučních chyb není syntakticky neobvyklých a mnoho nezvyklých konstrukcí neobsahuje chyby. Naopak pozdější přístupy měly velmi omezené použití vzhledem k předpokladu, že každá věta obsahuje nejvýše jedno nesprávně napsané slovo, každá chyba je způsobena pouze jednou bodovou změnou a defektní poměr (relativní množství chyb v textu) je známý [3].

Dalším a hlavním problémem, který vylučuje tradiční metody pro opravu překlepů pro použití k opravě dotazů zadávaných do vyhledavače, je fakt, že využívají pouze slovník daného jazyka. To zdaleka nepokrývá různorodost dotazů, které musejí dnešní vyhledavače zpracovávat, jelikož uživatelé často vyhledávají cizojazyčná slova a fráze nebo nově vzniklá slova, která se v běžném slovníku nenacházejí.

1.3. Oprava překlepů zadávaných do vyhledávače

Oprava překlepů dotazů zadávaných do vyhledávače, kterou se tato práce zabývá, má mnoho společného s tradiční korekcí chyb, ale také představuje další úkoly, na které je potřeba se zaměřit a zohlednit je při návrhu řešení. Četnost a závažnost pravopisných chyb je v tomto případě významně větší než v obyčejném psaném textu. Zhruba 10–15 % z webových dotazů obsahuje chyby [3, 5]. Platnost dotazu nemůže být typicky posouzena na základě vyhledávání ve slovníku nebo kontrolováním gramatiky. Jelikož jsou webové dotazy velmi krátké (v průměru méně než 3 slova), nelze použít ani techniky založené na širokých kontextových souvislostech. Spíše než z dobře formulovaných vět se většina dotazů skládá z jednoho nebo více pojmů. Mnohokrát také obsahují slova, která nejsou uvedena ve slovníku, jako jsou slova cizí nebo nově vytvořená.

Definovat platný webový dotaz je velmi obtížná věc, protože nemůžeme jednoduše použít existující slovník, jelikož každý den vzniká mnoho nových slov a pojmů (blog, shrek apod.) a bylo by složité nebo dokonce nemožné udržovat slovník stále aktuální. Používání velkého slovníku by mohlo mít navíc za následek několik

problémů, jako je záměna platných slov vedoucí k nesmyslné opravě korektního dotazu.

Jednou z alternativ, z které lze při opravě dotazů zadávaných do vyhledávače vycházet, je využití miliónů záznamů již provedených dotazů. Dalo by se tvrdit, že relevance určitého slova může být odvozena z jeho četnosti, s jakou ho uživatelé vyhledávají. Takový přístup má však několik omezení. Například by bylo chybné z logů dotazů jednoduše extrahovat všechna slova, jejichž četnost přesahuje určitou stanovenou hodnotou, a považovat je za platné. Některé oblíbené a často hledané výrazy se v chybném tvaru vyskytují s větší četností než správně psaná, ale méně hledaná slova. Hlavním úkolem je tedy zjistit, které ze záznamů již provedených dotazů jsou platné a které nikoliv, což není zase tak snadné. Na jednu stranu nám logy dotazů poskytují velké množství informací, ale na druhou stranu s nimi musíme zacházet opatrně, aby bylo jejich využití efektivní.

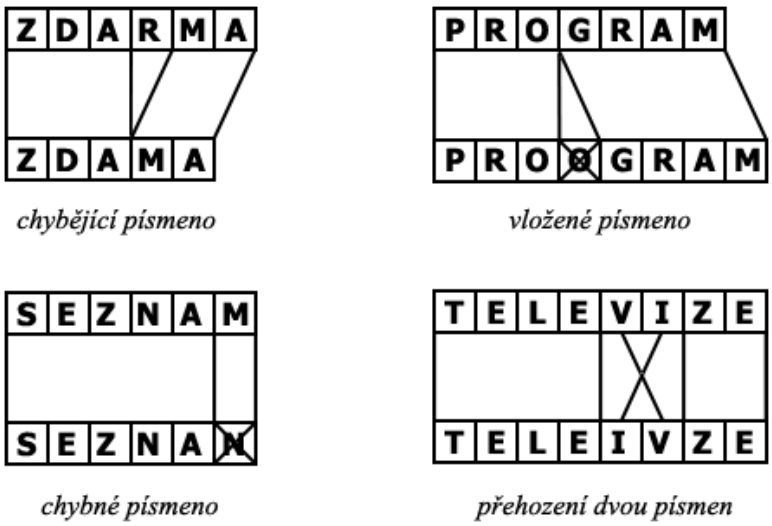
Klíčový požadavek kladený na opravu překlepů webových dotazů spočívá ve schopnosti algoritmu vybrat z nabízeného množství oprav tu nejlepší. Tradiční metody pro kontrolu pravopisu nabízejí seznam možných oprav, z kterých si uživatel musí sám vybrat, což není v případě internetových vyhledávačů žádoucí. Proto je důležité, aby byl algoritmus schopný vybrat automaticky nejlepší možnou opravu i za předpokladu, že při velmi špatně zadaném dotazu nenabídne opravu žádnou.

2. Teorie opravy překlepů

2.1. Druhy chyb

Chyby vzniklé v psaném textu můžeme rozdělit do dvou kategorií. První jsou *typografické chyby*, které vznikají nejčastěji stiskem nesprávné klávesy, stiskem dvou kláves, stiskem kláves v nesprávném pořadí a podobně. Častým důvodem bývá špatně nastavené rozložení klávesnice, z čehož vznikají problémy, jako je záměna písmen *z* a *y*, případně psaní číslic namísto českých znaků.

Druhou kategorii tvoří *fonetické chyby*, kdy slovo zní při vyslovení stejně, ale pravopis je špatný. Do této kategorie tedy patří *pravopisné chyby*, které jsou v českém jazyce celkem běžné. Nesprávné psaní měkkého a tvrdého *i* patří k častým prohřeškům, které vedou ke vzniku špatného dotazu. Podobných nástrah je v českém jazyce více, proto je potřeba je zohlednit při návrhu samotného algoritmu pro opravu překlepů. Fonetické chyby často také vznikají u slov cizích nebo odborných, kde uživatel nezná přesnou podobu slova a „píše ho, jak ho slyší“.



Obr. 2.1: Základní druhy překlepů

Z hlediska složitosti je obvykle těžší opravit fonetické chyby, pokud se zrovna nejedná o jednu z častých chyb (např. výše zmíněné psaní *i* a *y*), protože slovo v takovém případě většinou obsahuje větší množství elementárních chyb, které jsou zobrazeny na Obr. 1. Mezi ně patří již dříve zmíněné *vložení*, *smazání*, *nahrazení* a *přehození* písmen.

2.2. Metody pro opravu překlepů

Vývoj technik pro zjištění a následné opravení pravopisných chyb má za sebou velmi dlouhou historii, která sahá až do poloviny 20. století [3]. V dnešní době jsou nástroje pro opravu překlepů běžnou záležitostí a vyskytují se v mnoha programech a pokrývají většinu jazyků. Pro opravu bylo navrženo velké množství metod, mezi které patří editační vzdálenost, n-gramy, techniky založené na pravděpodobnosti, neuronové sítě, techniky založené na mluvnických pravidlech, techniky založené na podobnosti klíčů více slov nebo jejich kombinace [3]. Všechny tyto metody vycházejí ze stejné myšlenky, kterou je počítání vzdálenosti mezi nesprávně napsaným slovem a každým slovem nacházejícím se ve slovníku. Čím nižší tato vzdálenost je, tím výše je dané slovníkové slovo řazeno v seznamu možných oprav a tím vyšší je pravděpodobnost, že se jedná o opravu korektní.

2.3. Editační vzdálenost

Celkem jednoduchou metodou, která se používá pro opravu překlepů, je editační vzdálenost. Tuto vzdálenost představuje číslo, které odpovídá počtu elementárních operací potřebných pro transformaci jednoho slova v druhé. Analýzou pravopisných chyb ve velkých textových souborech bylo zjištěno, že velká většina nesprávně psaných slov (80–95%) [5] vzniká právě jednou ze čtyř chyb zobrazených na obrázku 2.1. Proto by základní editační operace měly odpovídat právě těmto čtyřem chybám. Možní kandidáti na opravu by se pak měli lišit od původního slova v co nejmenším počtu těchto operací. Pro zjištění editační vzdálenosti je možné použít několik metod, které byly v minulosti definovány. Mezi nejznámější a nejpoužívanější patří Hammingova vzdálenost a Levenshteinova vzdálenost.

Typická je také dodatečná heuristická analýza, která tvoří doplněk právě pro techniky založené na editační vzdálenosti. Například v případě typografických chyb je důležité vzít v úvahu rozvržení klávesnice, protože mnohem častěji vznikají záměny písmen v důsledku stisku klávesy, která se nachází v těsné blízkosti „správné“ klávesy.

2.3.1. Hammingova vzdálenost

Hammingova vzdálenost (Hamming distance), která byla formulována Richardem Hammingem roku 1950, udává počet pozic, na kterých se liší dva řetězce stejné délky. Jedná se tedy o počet záměn, které je potřeba provést, abychom

převedli jeden řetězec na druhý. Tato vzdálenost může být určena jak pro znaková slova tak pro binární slova, jak je vidět z následujícího příkladu. Pozice, na kterých se daná slova liší, jsou zvýrazněna tučným fontem.

- 10**1**1**1**01 → 10**0**1**0**01 HV = 2
- 2**1**73**8**96 → 2**2**3**3**796 HV = 3
- zda**rm**a → zda**tn**a HV = 2

2.3.2. Levenshteinova vzdálenost

Levenshteinova vzdálenost (Levenshtein distance) udává minimální počet operací, které je potřeba provést pro transformaci jednoho řetězce na druhý. Mezi tyto operace patří vložení, smazaní a náhrada jednoho znaku. Představuje tak zobecnění Hammingovy vzdálenosti, která uvažuje pouze operaci náhrady znaku, a používá se zejména pro znaková slova. Následující příklad ukazuje výpočet Levenshteinovy vzdálenosti po jednotlivých krocích pro chybně napsané slovo *atpbuss* a slovníkovou variantu *autobus*.

1. at**p**buss → at**o**buss (náhrada „p“ za „o“)
2. atobus**s** → atobus (smazání zdvojeného „s“)
3. atobus → a**u**tobus (vložení „u“)
výsledná LV = 3

Existují také další zobecnění Levenshteinovy vzdálenosti a jednou z nich je Damerau-Levenshteinova vzdálenost, která je v této práci použita jako základ algoritmu pro opravu překlepů. V tomto případě se soubor základních editačních operací, které je možné s řetězcem provádět, rozšiřuje vzájemné přehození dvou znaků, které by se jinak považovalo za dvě operace nahrazení znaku. Následuje ukázkový příklad výpočtu pro slova *hooroskpo* a *horoskop*.

1. ho**o**roskpo → horoskpo (smazaní zdvojeného „o“)
2. horosk**po** → horosk**op** (přehození „p“ a „o“)
výsledná DLV = 2

2.4. Techniky založené na podobnosti klíčů

Techniky vycházející z podobnosti klíčů jsou založeny na transformaci slov do podobnostních klíčů, které vyjadřují charakteristiku daného slova. Testované slovo a slova nacházející se ve slovníku jsou převedena do podoby klíče. Následně jsou všechna slova, která mají stejný klíč jako testované slovo, vrácena jako kandidáti na opravu.

2.4.1. Soundex

Příkladem této metody je oblíbený algoritmus Soundex, který byl navržený hlavně pro opravu fonetických chyb a vychází z anglické výslovnosti slov. Byl vyvinut a patentován již v roce 1918 a za jeho vznikem stojí Robert Russell a Margaret Odell. Postupem času vzniklo několik odvozených variant jako například Reverse Soundex, Celko Improved Soundex nebo Daitch-Mokotoff Soundex.

Funkci je předáno slovo a výstupem je jeho čtyřmístný kód tvořený jedním písmenem, které odpovídá počátečnímu písmenu daného slova, a třemi číslicemi (např. slovo *program* má kód P626). Pokud je stejný klíč vrácený i pro jiné slovo, je velmi pravděpodobné, že se jedná a možnou opravu pro testovaný výraz.

2.4.2. Metaphone

Dalším algoritmem založeným na transformaci slov do klíčů na základě jejich výslovnosti je *Metaphone*. Na rozdíl od Soundex, který analyzuje slovo po jednotlivých písmenech, je tento algoritmus navržený tak, aby vycházel jak z hlásek tak dvojhlásek (slabik) a to podle určitého souboru pravidel, která charakterizují skládání písmen do větších skupin. Rozdílný je také výsledný klíč, který může mít u této metody různou délku (slovo *program* má klíč PRKRM). Metaphone má ovšem stejné omezení jako předchozí Soundex a to je možnost využití jen pro anglický jazyk, jelikož algoritmus vychází právě z anglické výslovnosti. Bylo by samozřejmě možné vytvořit podobný algoritmus i pro ostatní jazyky podle jejich pravidel výslovnosti, ale takové varianty k dispozici nejsou.

2.5. N-gramy

V tradičním přístupu pro opravu překlepů je důležitou součástí využití kontextu, pomocí kterého je možné zjistit nesprávné použití bezchybně psaného slova, například jeho špatné umístění ve větě. Přestože dotazy zadávané uživateli do

vyhledávače neobsahují většinou více než tři slova, není možné použití kontextu při opravě úplně vypustit. Algoritmus založený pouze na opravě jednotlivých separovaných slov by v mnoha případech neposkytoval optimální návrhy na opravu dotazu jako celku.

Základem pro kontextově závislou opravu překlepů je využití tzv. n-gramů. N-gram je vybraná posloupnost (podposloupnost) *n* položek z dané posloupnosti. Položky mohou reprezentovat písmena, slova nebo jiné elementy v závislosti na konkrétním použití. N-gram o velikosti 1 se nazývá *unigram,* o velikosti 2 *bigram* a o velikosti 3 *trigram*. V této práci jsou použity unigramy a bigramy.

N-gramy lze s úspěchem využít při modelování souvislostí, která v jazyce vznikají. Lze tak na základě statistických vlastností jednotlivých bigramů a pomocí Bayesovy věty předvídat, které slovo bude s největší pravděpodobností následovat po jiném. Při tom se předpokládá, že slovo závisí pouze na *n* předchozích slovech, takže čím vyšší je *n*, tím vyšší je přesnost odhadu.

3. Analýza záznamů dotazů

Základem v této práci navrženého algoritmu pro opravu překlepů jsou záznamy již provedených dotazů na portálu Seznam.cz. Tyto údaje poskytují mnoho informací, které vedou k navržení mnohem efektivnějšího algoritmu, než pokud bychom vycházeli pouze ze slovníku. Hlavní výhodou je znalost četností, s jakou uživatelé jednotlivá slova vyhledávají. Jelikož je soubor dotazů dostatečně velký, je možné ze získaných četností vycházet a v algoritmu je použít jako hlavní kritérium při výběru nejlepší opravy v případě, že je nalezeno více vhodných kandidátů.

Druhou výhodou, kterou přináší znalost uživatelských dotazů, je možnost opravovat i neslovníkové výrazy jako jsou cizí slova, nespisovné výrazy nebo například adresy internetových stránek, které uživatelé vyhledávají překvapivě často. V tomto případě je ovšem nutné pečlivě vybírat, zda je výraz vytažený ze záznamů validní a tím pádem je možné ho použít jako kandidáta na opravu. K tomu je také možné použít získané četnosti.

3.1. Zpracování záznamů

3.1.1. Parsování souborů

Seznam.cz poskytnul záznamy (logy) dotazů, které uživatelé zadávali do vyhledávače v průběhu prvních 14 dní roku 2008. Jedná se celkem o téměř **170 miliónů** samostatných dotazu, což poskytuje dostatečný statistický soubor, ze kterého lze při opravě překlepů vycházet.

```
GET /?q=ct+24+online&mod=f&sug=1 HTTP/1.0
GET /?q=leann+rimes+amazing+grace+lyrics&mod=g HTTP/1.0
GET /?q=&mod=f HTTP/1.0
GET /?q=lcd+monitory&mod=f&sug=1 HTTP/1.0
GET /?q=mp3s.nadruhou.net&mod=f&sug=1 HTTP/1.0
GET /?q=zirafa&nocache=1&monitoring=1 HTTP/1.0
```

Obr. 3.1: Ukázka logu

Nejdříve bylo ovšem nutné surové logy nějakým způsobem zpracovat. K tomu jsem naprogramoval parser, který dostal jako vstupní data zmíněné soubory s logy a výstupem byla požadovaná statistika unigramů a bigramů. Podoba záznamů, které tvořily jednotlivé soubory, je vidět na obrázku 3.1. Každý soubor obsahoval

záznamy logů za uplynulých 24 hodin, takže souborů bylo celkem 14. Jednotlivé řádky představují dotazy uživatelů doplněné o několik informací, které jsou pro tuto práci nepodstatné. Základem tedy bylo získat vlastní text dotazu, s kterým pak bylo možné dále pracovat.

Jednotlivé soubory byly zpracovávány sekvenčně řádek po řádku a nejdříve bylo nutné odstranit z každého řádku nepotřebné doplňkové informace. Jelikož se jedná o webové dotazy, které jsou zakódované do tvaru bezpečného pro přenos v URL, bylo následně potřeba každý řetězec dekódovat pomocí standardní funkce URL decode. Výsledkem byl dotaz ve tvaru, v jakém jej uživatel zadal do vyhledávače. Dalším krokem bylo každý dotaz rozdělit na jednotlivé n-gramy, v případě této práce na unigramy a bigramy, a takto získané prvky nějakým způsobem uchovávat, aby bylo možné v průběhu zpracování jednotlivých souborů aktualizovat jejich četnosti.

Při výběru vhodné datové struktury jsem vycházel ze dvou základních požadavků. Prvním je přiměřená paměťová náročnost, jelikož se najednou zpracovává velké množství dat, které je potřeba udržovat v paměti. Druhým požadavkem je rychlé vyhledávání v dané struktuře, protože je potřeba zjistit, zda se získaný n-gram již v paměti nenachází.

3.1.2. Binární vyhledávací strom

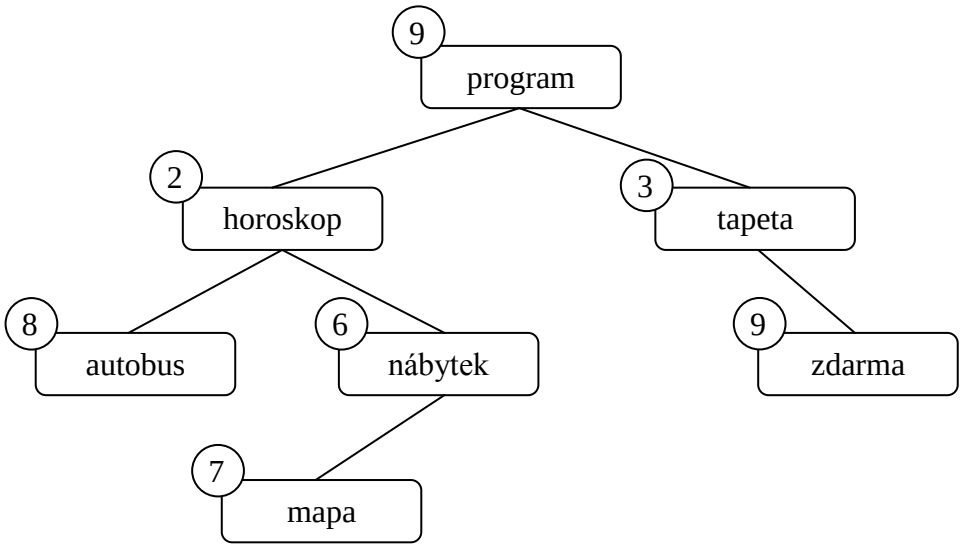
Jako datovou strukturu pro uchování a zpracování n-gramů jsem nakonec použil binární vyhledávací strom (BST), protože nejlépe splňuje výše stanovená kritéria. Dalšími kandidáty byly například ternární vyhledávací strom nebo digitální vyhledávací strom. Jejich využití ovšem znemožňují velké nároky na paměť.

Binární vyhledávací strom je datová struktura založená na binárním stromu, v němž jsou jednotlivé prvky (uzly) uspořádány tak, aby v tomto stromu bylo možné rychle vyhledávat danou hodnotu. Hlavní výhodou BST je tedy vysoká efektivita vyhledávání hodnot v nich [2]. To zajišťují tyto čtyři vlastnosti:

- Jedná se o binární strom, každý uzel tedy má nanejvýš dva potomky − levého a pravého.
- Každému uzlu je přiřazen určitý klíč. Podle hodnot těchto klíčů jsou uzly uspořádány.
- Levý podstrom uzlu obsahuje pouze klíče menší než je klíč tohoto uzlu.
- Pravý podstrom uzlu obsahuje pouze klíče větší než je klíč tohoto uzlu.

Všechny operace prováděné na binárním stromě, mezi které patří přidání nového uzlu, smazání uzlu, prohledávání stromu, procházení stromu a některé další, opakovaně porovnávají hodnoty klíčů. Může se jednat o triviální porovnání dvou čísel či řetězců, ale také o složitější podprogram, pokud úlohu klíče hraje kombinace několika datových položek uzlu. Kromě klíče uzel zpravidla nese další datové položky, které tvoří vlastní obsah datové struktury.

Pro potřeby této práce jsem využil BST, jehož klíče tvořily jednotlivé n-gramy získané ze záznamů dotazů. Každý uzel si kromě tohoto klíče nesl i záznam o aktuální četnosti, to znamená číslo, které udává, kolikrát se daný n-gram nacházel ve zpracovaných souborech. Výsledná struktura BST je vidět na obrázku 3.2.



Obr. 3.2: BST obsahující sedm slov (unigramů)

Postup zpracování každého vyextrahovaného n-gramu byl následující. Nejdříve bylo nutné pomocí prohledávání stromu zjistit, zda se daný prvek již v BST nenachází. Pokud ne, n-gram se vložil do BST pomocí standardní funkce pro vytvoření nového uzlu a přiřadila se mu četnost „1“. V opačném případě se inkrementovala hodnota četnosti o jedničku.

Výše zmíněný postup se aplikoval na všechny soubory se záznamy uživatelských dotazů. Po jeho skončení obsahoval jeden binární strom statistiku všech unigramů a druhý statistiku všech bigramů. Následně bylo potřeba oba stromy projít a vybrat z nich pouze záznamy, které měly četnost nad určitou pevně stanovenou hranicí, protože n-gramy s nízkou četností výskytu v dotazech není

možné pro opravu překlepů použít. Většinou se totiž jedná právě o chybně zadané dotazy, které se mockrát neopakují. Tímto způsobem je tak možné odfiltrovat velké množství nepotřebných záznamů a tím snížit výsledný statistický soubor, který by byl jinak neúměrně velký.

N-gramy vybrané z BST byly seřazeny abecedně, což vychází ze samotné podstaty stromu. Proto bylo potřeba je následně seřadit podle četnosti, protože takto setříděná statistika se dala s úspěchem využít v samotném algoritmu pro opravu překlepů. Nakonec jen zbývalo všechna získaná data uložit do souboru, aby je bylo možné použít při vlastní opravě.

3.2. Statistiky dotazů

Na základě zpracovaných dotazů jsem vytvořil několik statistik, které jsou přehledně zobrazeny v následujících tabulkách. Tabulka 3.1 obsahuje přehled nejčastěji hledaných unigramů a bigramů a jsou z ní patrné dvě věci. Na pátém místě figuruje řetězec „2008“, což je ovlivněno tím, že se jednalo o záznamy uživatelských dotazů za prvních 14 dní roku 2008. Z toho vyplývá, že jsou hledané dotazy ovlivněny časovým obdobím, například ročním obdobím, ve kterém jsou uskutečněny, protože lidé mají potřebu získat jiné informace v létě a jiné v zimě. Vzhledem k této skutečnosti by bylo optimální zpracovat statistiku dotazů, které pokrývají větší časové období, nejlépe jsou rovnoměrně rozloženy přes celý rok. Pro potřeby této práce ovšem stačí dotazy za zmíněné období, protože často hledané dotazy se s časem nemění, pokud nebere v úvahu opravdu široké časové rozpětí (několik let). To je také vidět z unigramů, které jsou v tabulce na prvním až čtvrtém místě.

pořadí	unigramy		bigramy	
	řetězec	**abs. četnost**	**řetězec**	**abs. četnost**
1.	mp3	2 062 727	stazeni zdarma	365 127
2.	zdarma	1 761 662	stažení zdarma	268 846
3.	hry	1 247 273	sms zdarma	200 700
4.	download	893 303	mp3 stazeni	135 860
5.	2008	840 227	jízdní řády	133 840

Tab. 3.1: Statistika dotazů (všechna slova)

Druhý poznatek, který je možné z tabulky 3.1 získat, je vidět ve sloupci bigramů. Na prvním a druhém místě se umístily vlastně stejné řetězce. Jediný rozdíl je pouze v psaní diakritiky, kterou mnoho lidí kvůli zjednodušení při vyhledávání nepoužívá. Toto by se mělo při samotném návrhu algoritmu také zohlednit.

Následující dvě tabulky ukazují rozložení délky dotazů podle počtu slov, které je tvoří. V případě tabulky 3.2 jsou zahrnuta všechna slova bez ohledu na jejich délku. Naopak tabulka 3.3 vychází pouze ze slov o minimální délce tří znaků, takže se do statistik nezapočítávají velmi krátký výrazy jako předložky a podobně. Také většina termínů, které tvoří základ dotazů a podle kterých se následně vrací výsledky vyhledávaní, nemá délku kratší než tři znaky, a proto má tato tabulka větší vypovídající hodnotu.

Jak je vidět ze sloupce relativní četnosti, tvoří dotazy zadávané do vyhledávače nejčastěji jedno (33,51 %) nebo dvě slova (37,20 %). Každý pátý dotaz (19,62 %) pak obsahuje slova tři. Pokud se sečtou všechny tyto četnosti, dostaneme hodnotu 90,62 %, která je vidět ve sloupci relativní kumulované četnosti. Z toho vyplývá, že 90 % uživatelských dotazů tvoří maximálně tři slova. Průměrná hodnota délky dotazu je pak 2,32 slova. Proto také navržený algoritmus na opravu překlepů vychází pouze z unigramů a bigramů.

délka dotazu [počet slov]	četnost		kumulovaná četnost	
	absolutní	**relativní**	**absolutní**	**relativní**
1	53 153 625	31,41%	53 153 625	31,41%
2	54 974 567	32,48%	108 128 192	63,89%
3	33 934 615	20,05%	142 062 807	83,94%
4	16 023 907	9,47%	158 086 714	93,41%
5	6 454 244	3,81%	164 540 958	97,22%
6	2 763 288	1,63%	167 304 246	98,86%
7	1 085 057	0,64%	168 389 303	99,50%
8	449 244	0,27%	168 838 547	99,76%
9	186 805	0,11%	169 025 352	99,87%
10	99 790	0,06%	169 125 142	99,93%
11-20	109 256	0,06%	169 234 398	100,00%
21+	7 474	0,00%	169 241 872	100,00%
celkem	**169 241 872**	**100,00%**	-	-

Tab. 3.2: Statistika dotazů (všechna slova)

délka dotazu [počet slov]	četnost		kumulovaná četnost	
	absolutní	relativní	absolutní	relativní
1	56 705 524	33,51 %	56 705 524	33,62 %
2	62 950 222	37,20 %	119 655 746	70,94 %
3	33 198 342	19,62 %	152 854 088	90,62 %
4	11 038 651	6,52 %	163 892 739	97,16 %
5	3 301 031	1,95 %	167 193 770	99,12 %
6	993 103	0,59 %	168 186 873	99,71 %
7	294 188	0,17 %	168 481 061	99,88 %
8	115 214	0,07 %	168 596 275	99,95 %
9	38 746	0,02 %	168 635 021	99,97 %
10	17 660	0,01 %	168 652 681	99,98 %
11-20	24 595	0,01 %	168 677 276	100,00 %
21+	3 972	0,00 %	168 681 248	100,00 %
celkem	**168 681 248**	**100,00%**	**-**	**-**

Tab. 3.3: Statistika dotazů (jen slova o min. délce 3 znaky)

Ze záznamů dotazů je také zřejmé, že uživatelé často vyhledávají přímo internetové adresy stránek. Takové dotazy pak většinou obsahují předponu „www“ nebo příponu odpovídající kořenové doméně (cz, com, net, …). Před samotným vyhodnocením dotazu je pak vhodné odstranit nejméně prefix „www“ a opravu hledat jen pro vlastní název stránky.

4. Návrh algoritmu pro opravu překlepů

4.1. Požadavky na algoritmus

Na algoritmy realizující opravu překlepů zadávaných do vyhledávače jsou kladny mnohem vetší nároky než na běžné spell checkery, které se používají například v textových editorech nebo jiných programech pro opravu psaného textu. Hlavní rozdíl se týká času, který má daný algoritmus k dispozici pro navržení opravy. Druhým rozdílem je pak skutečnost, že při korekci webových dotazů je nutné vybrat z většího množství kandidátů tu nejlepší opravu, která je následně nabídnuta uživateli.

4.1.1. Časová náročnost

Tradiční řešení pro opravu překlepů nejsou časem nějak extrémně limitovány. K hledání možných chyb a následné korekci dochází průběžně při editaci textu a uživatele toto nijak nezdržuje. U opravy webových dotazů je to však jiné. Přestože se výpočetní výkon serverových stanic neustále zvyšuje obrovským tempem, je potřeba při návrhu algoritmu klást velký důraz na jeho časovou náročnost. Internetové vyhledávače musejí v jednom okamžiku zpracovat velké množství dotazů (Seznam.cz vyřizuje stovky dotazů za sekundu), a proto je čas pro vyřízení každého jednoho z nich kritický. Algoritmus pro opravu překlepů pak představuje další zdržení, které je nutné připočítat k celkovému času potřebnému k vyhodnocení dotazu, a proto musí být jeho implementace co nejrychlejší. Je nutné také omezit maximální dobu pro hledání opravy a to i za cenu, že nebude vhodná korekce nalezena a nabídnuta uživateli. Jinak by se mohlo stát, že bude uživatel čekat na výsledky vyhledávání relativně dlouhou dobu a to je pro internetové vyhledávače nepřípustné.

4.1.2. Výběr nejlepší opravy

Druhým rozdílem, kterým se odlišuje algoritmus navrhovaný v této práci od běžných spell checkerů, je způsob, jakým se zpracuje seznam možných kandidáti na opravu, protože ve většině případů je nalezen více než jeden. Tradiční přístup používaný například v textových editorech v takovém případě nabídne uživateli celý seznam kandidátů a ten si z něj vybere tu správnou opravu. Při opravě překlepů dotazů je ovšem nutné vybrat pouze jednou variantu opravy, které je následně zobrazena uživateli společně s výsledky vyhledávání. V algoritmu je proto nutné řešit výběr nejvhodnějšího kandidáta.

4.2. Výchozí data algoritmu

Základ algoritmu tvoří několik druhů dat, ze kterých se vychází při vlastní opravě dotazů. Jedna část je tvořena slovníkovými daty, která vymezují korektní slovní tvary pro daný jazyk. Druhou část tvoří statistika unigramů a bigramů, která vychází ze záznamů provedených dotazů. Jejich zpracování je popsáno v části 3. Posledním zdrojem informací je pak seznam stalých (fixních) oprav.

4.2.1. Ověřený slovník - Hunspell

Slovníková data představují soubor slov, která lze automaticky považovat za správná. Hlavní využití slovníku je při počáteční kontrole, zda je vůbec nutné zadaný dotaz opravovat. Pokud je slovo nalezeno ve slovníku, je vyhodnoceno jako korektní a dále se neopravuje.

Samotná realizace slovníku a jeho udržování v paměti je provedeno pomocí knihovny Hunspell. Jedná se o spell checker založený na podobné knihovně MySpell. Projekt je vydávaný pod GNU LGPL licencí, a proto je ho možné použít i v komerčních aplikacích, které spadají pod jinou licenci. Jelikož se jedná o morfologický analyzátor [11], hodí se Hunspell pro jazyky s velmi bohatým tvaroslovím a velkým množstvím složenin, a proto dovoluje kontrolovat slova v různých tvarech, tzn. podstatná jména ve všech pádech a osobách nebo slovesa ve všech časech i osobách, a proto se výborně hodí právě pro český jazyk. Tento spell checker se používá pro kontrolu pravopisu například v kancelářském balíku OpenOffice.org nebo v produktech společnosti Mozilla jako jsou Firefox 3 a Thunderbird.

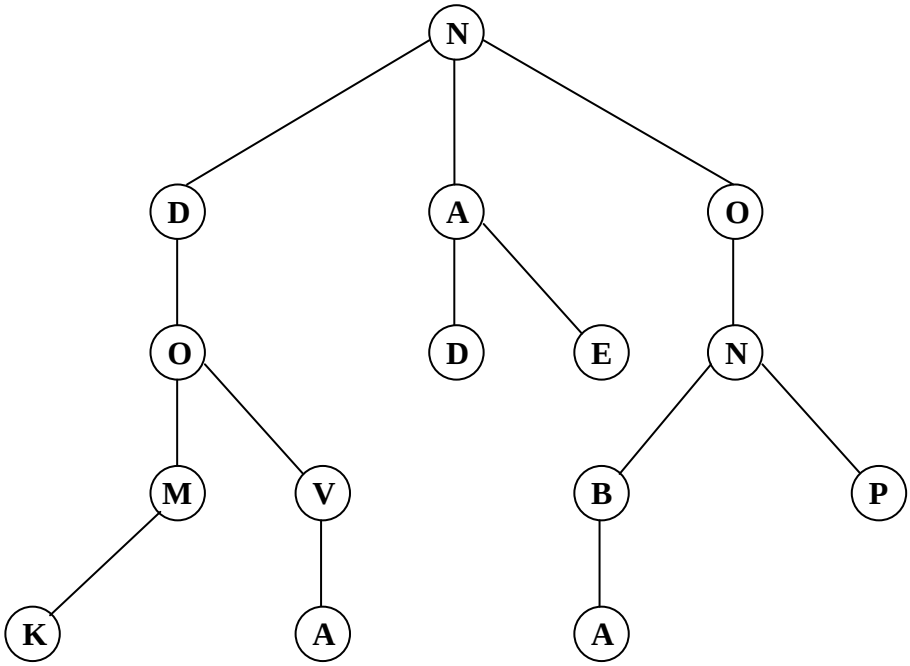
Knihovna je psaná v C++, a proto bylo její použití v algoritmu velmi snadné, i když se omezilo jen na kontrolu, zda se daný výraz ve slovníku vyskytuje či nikoliv. Hunspell jinak samozřejmě poskytuje funkce jako plnohodnotný nástroj pro opravu překlepů. Jeho případné využití pro přímou opravu překlepů v dotazech však znemožňuje dlouhý čas potřebný v některých případech pro nalezení opravy (až 0.5s) a také skutečnost, že umí opravovat pouze slovníková slova, což je pro různorodé dotazy zadávané do vyhledávače nedostatečné.

V algoritmu jsem primárně využíval český slovník, ale jelikož se v hledaných dotazech často vyskytují cizojazyčná slova a to hlavně anglická, využil jsem pro kontrolu dotazů také slovník anglický.

4.2.2. Statistika dotazů – ternární vyhledávací strom

Jako základní datovou strukturu pro udržování získaných n-gramů a jejich četností v paměti jsem použil ternární vyhledávací strom (TST). Jedná se o strukturu, která kombinuje rychlost vyhledávání digitálního stromu a navíc má nižší paměťové nároky podobné binárnímu stromu [1].

TST je typ stromu, ve kterém má každý uzel tři potomky. Pokud se takový strom navíc používá pro ukládání asociativního pole, kde klíči jsou řetězce, nazývá se „trie“. Při ukládání řetězců pak každý uzel představuje jeden znak. Příklad takového stromu je na obrázku 4.1.



Obr. 4.1: TST obsahující 10 výrazů:
do, dok, dom, dva, na, nad, ne, oba, on, op

Výhodou TST je rychlé vyhledávání, které je v nejhorším případě rovno *O(log(n) + k)*, kde *n* je počet řetězců uložených ve stromu a *k* je délka hledaného řetězce. Při samotném hledání se strom prochází od kořene a vždy se porovnává jeden znak hledaného řetězce s aktuálním uzlem. Pokud je hledaný znak menší než hodnota uzlu, pokračuje se levou větví, pokud je hodnota větší, pokračuje se pravou větví. Když se znaky rovnají, pokračuje se prostřední větví a přejde se na hledání dalšího znaku v řetězci. Toto vyhledávání je v mnoha případech rychlejší než při

použití hashovací tabulky (např. nižší čas při neúspěšném hledání) a navíc TST poskytuje několik užitečných funkcí přímo ze své podstaty. Mezi ně patří například hledání všech řetězců, které mají danou předponu nebo příponu.

4.2.3. Stálé opravy

Stálé (fixní) opravy tvoří poslední část statistického souboru, kterou využívá algoritmus v první fázi opravy překlepů. Jedná se o seznam dvojic chyba – oprava, které přesně definují opravy některých špatně zadávaných dotazů. V tomto případě jde o nejčastější chyby a jejich opravy, které uživatele zadávají do vyhledávače. Využití těchto pevně daných dvojic je v praxi velmi efektivní, protože proces korekce je v tomto případě stoprocentně přesný a také velmi rychlý.

V případě této práce jsem použil soubor 350 stálých oprav, které jsem získal ze statistiky dotazů. K jejich vygenerování jsem použil navržený algoritmus pro opravu překlepů, který jsem nechal opravit určité množství nejčastějších dotazů, mezi nimiž se nacházejí i nejčastější překlepy, které uživatelé zadávají do vyhledávače. Tento postup samozřejmě navrhl i velké množství nesprávných oprav, a proto bylo nutné následně všechny ručně projít a vybrat jen ty správné případy.

"youtube.cz"	→	*"youtube.com"*
"nokie"	→	*"nokia"*
"sonyericson"	→	*"sony ericsson"*
"donwload"	→	*"download"*
"maunfield"	→	*"mountfield"*
"vikipedie"	→	*"wikipedia"*

Obr. 4.2: Ukázka souboru se stálými opravami

Fixní opravy je potřeba při inicializaci algoritmu načíst do vhodné struktury z externího souboru. K jejich udržování v paměti jsem využil stejně jako v případě statistiky n-gramů ternární vyhledávací strom, který poskytuje dostatečnou rychlost vyhledávání.

4.3. Celková stavba algoritmu

Hlavní a jedinou funkcí algoritmu je převzít řetězec, který představuje uživatelský dotaz zadaný do vyhledávače, a pokusit se pro něj navrhnout opravu, případně ho shledat jako korektní a opravu nenabízet. Algoritmus ve skutečnosti nezkoumá, zda dotaz potřebuje opravu, ale v případě, že žádnou vhodnou nenalezne, je původní dotaz považován za správný.

Samotný algoritmus pro opravu překlepů se skládá ze dvou logických částí. První část hledá vhodné kandidáty na opravu, mezi které se automaticky počítá i původní dotaz, pokud není nalezen ve slovníku. Druhá část má poté za úkol nalézt nejvhodnější opravu, která je následně nabídnuta uživateli jako možná oprava. Celý průběh algoritmu je zobrazen pomocí vývojového diagramu na obrázku 4.3.

4.4. Přípravná část

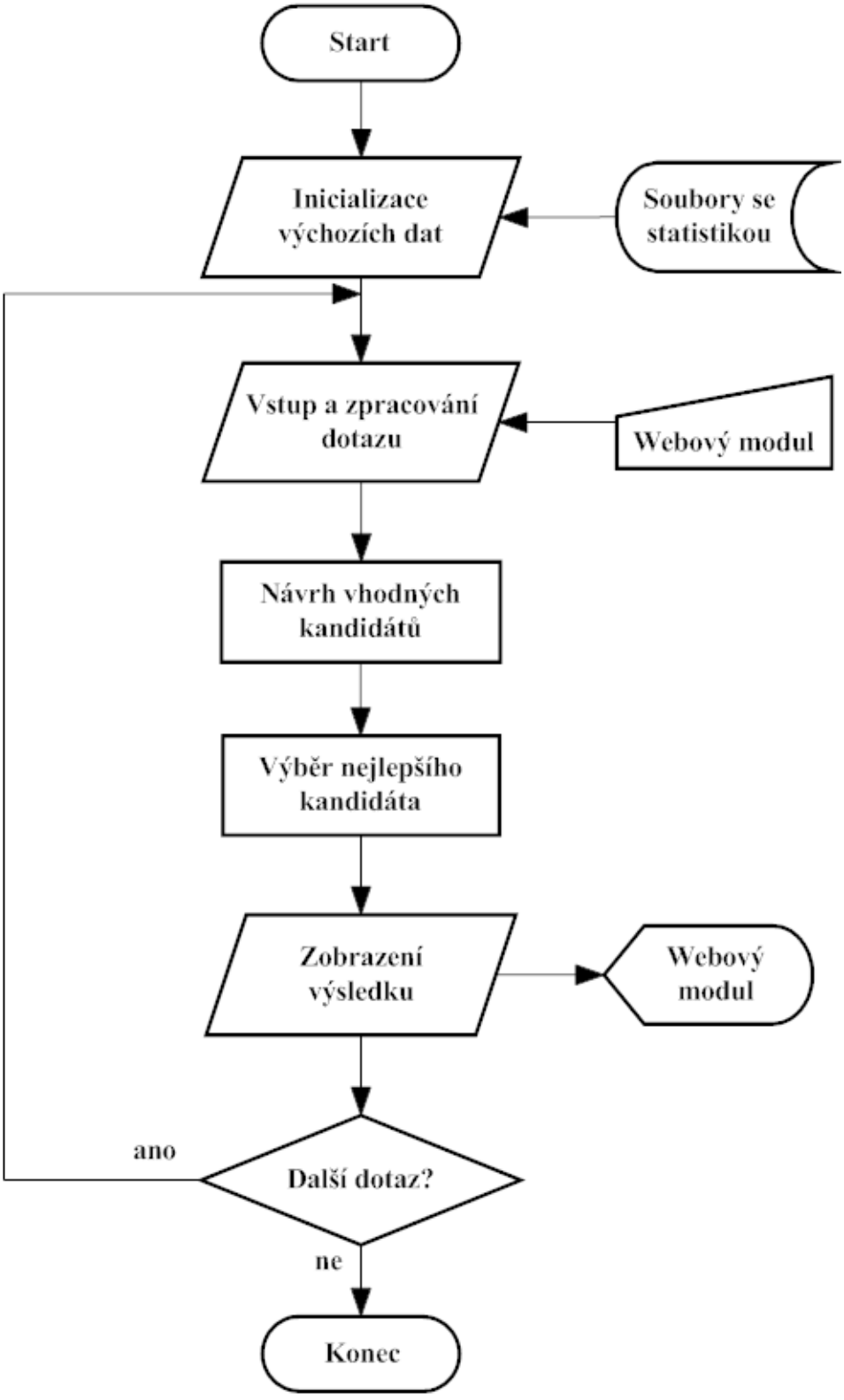
4.4.1. Inicializace výchozích dat

Před samotnou opravou dotazů je nutné provést inicializaci všech dat, ze kterých algoritmus vychází. Jedná se o tři hlavní části popsané v bodě 4.2. Ať už se jedná o Hunspell, statistiku dotazů nebo fixní opravy, jsou všechna data uložena v souborech. Proto je nutné nejdříve vytvořit potřebné datové struktury a následně je naplnit daty načtenými z těchto souborů. Tato operace se provádějí pouze jednou po spuštění programu (algoritmu) a poté se využívá k samotné opravě dotazů.

4.4.2. Vstup a zpracování dotazu

Po úspěšně dokončené inicializaci je algoritmus připraven převzít vstupní dotaz, který je následně zpracován. V případě skutečného nasazení bude dotaz předán pomocí webového serveru nebo jiné technologie přímo z webového modulu, kam uživatel svůj vstup zadá. Pro účely prototypové aplikace je dotaz zadáván prostřednictvím standardního vstupu přímo z klávesnice.

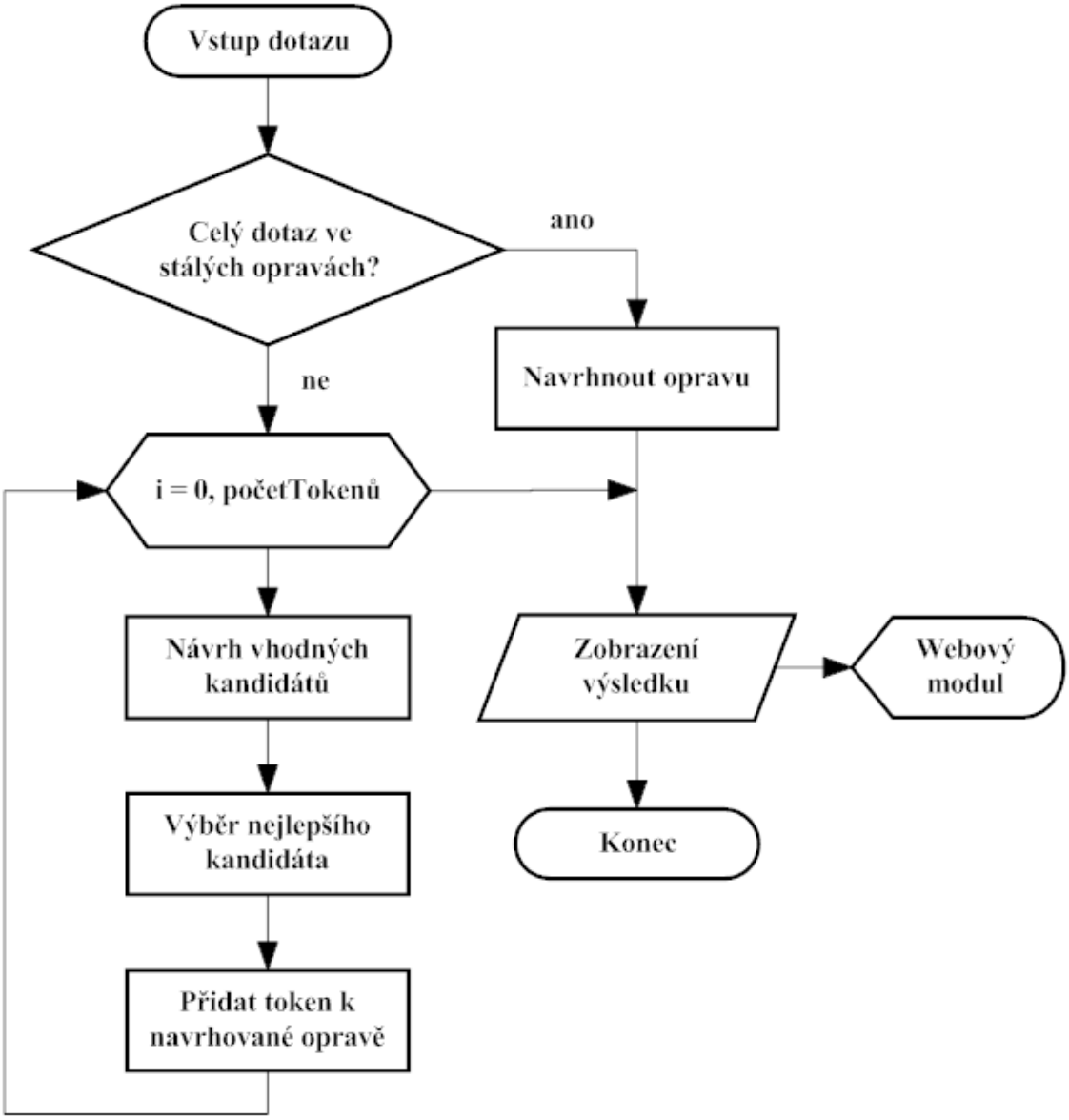
Získaný řetězec je následně nutné určitým způsobem zpracovat. Nejprve se převede na malá písmena a poté se rozdělí na jednotlivé tokeny, kterým ve většině případů odpovídají jednotlivá slova dotazu. Standardním oddělovačem je tedy znak mezery, případně některé další znaky jako tečka nebo pomlčka. Navržený algoritmus používá dělení pouze pomocí mezery.



Obr. 4.3: Grafické znázornění celého algoritmu

4.5. Návrh vhodných kandidátů

Návrh kandidátu na opravu je první ze dvou hlavních části algoritmu. Úkolem je co možná nejpřesněji nalézt možné varianty opravy vstupního řetězce a ty zařadit do seznamu kandidátů. V některých specifických případech je dokonce možné nalezenou opravu automaticky považovat za korektní a další již nehledat. V tomto případě se vůbec nevykoná druhá část algoritmu, tedy hledání nejvhodnější opravy ze seznamu kandidátů.



Obr. 4.4: Grafické znázornění průběhu opravy

Proces hledání vhodných kandidátů se aplikuje na každý vyhovující token dotazu zvlášť. Jedinou výjimku tvoří první krok, kde dochází k využití metody stálých oprav na celý dotaz. Samotný proces hledání se pak provádí pomocí různých postupů a technik, jejichž sled a průběh je popsaný v následujících odstavcích.

4.5.1. Kontrola vstupního tokenu

Než je možné přistoupit k vlastnímu hledání kandidátu na opravu, je nutné zkontrolovat, zda opravovaný řetězec splňuje dvě podmínky. První podmínka je minimální délka tokenu, která nesmí být menší než tři znaky. Jedno a dvou písmenné řetězce algoritmus neopravuje a rovnou je shledává jako korektní. Ve většině případů se jedná o nevýznamné části dotazu, na kterých výsledek vyhledávání nezaleží. Jde například o různé předložky, zájmena a podobně, jejichž opravou by docházelo k velkému množství chybných návrhů. Druhou podmínku je, že opravovaným řetězcem nesmí obsahovat číslo nebo jiný číselný údaj jako například datum, čas, IP adresa a některé další. Pokud token obě podmínky splňuje, je možné přikročit k vlastnímu hledání kandidátů. V opačném případě se neopravuje a pokračuje následujícím tokenem.

4.5.2. Použití stálých oprav

Jako první se na opravovaný token použije metoda fixních oprav. Nejprve se zkontroluje, zda se nenachází mezi opravami, na které se opravuje. To znamená, že jde o řetězec, který tvoří korektní tvar některého často zadávaného dotazu s překlepem. Pokud je řetězec v tomto souboru nalezen, je zřejmé, že se jedná o token ve správném tvaru a dále se neopravuje.

V druhém kroku se zkontroluje, zda se nejedná přímo o jednu z těchto častých chyb. Pokud ano, je řetězec opraven na korektní tvar.

4.5.3. Bigramová část

Druhým krokem je využití bigramů získaných z logů dotazů, ale samozřejmě jen v případě, že se dotaz skládá ze dvou nebo více tokenů. Pokud aktuálně opravovaný token není poslední v celém dotazu, je možné přikročit k bigramové části. Celý průběh je následující:

1. nastaví se maximální hodnota pro editační vzdálenost (MAX_LIMIT), aktuální LIMIT se nastaví na 1

2. vstupní bigram se vyhledá ve statistice bigramů, pokud je nalezen uloží se jeho četnost
3. for I = 0 to POCET_BIGRAMU - postupně se prochází pole bigramů seřazené sestupně podle četnosti
4. pokud se nerovnají první tokeny bigramů (aktuálně vybraný z pole a vstupní), pokračuje se dalším bigramem
5. pokud se délka druhých tokenů liší o více jak LIMIT znaků, pokračuje se dalším bigramem
6. spočítá se editační vzdálenost (Damerau-Levenshteinova vzdálenost) mezi druhými tokeny, pokud je větší než LIMIT, pokračuje se dalším bigramem
7. pokud je četnost aktuálního bigramu X krát větší než vstupního, vezme se tento aktuální jako platná oprava → konec; jinak se pokračuje dalším bigramem
8. inkrementuje se LIMIT, pokud není větší než MAX_LIMIT, pokračuje se krokem 3 a celé pole se znovu projde; jinak není oprava nalezena → konec

Uvedený postup se opakuje ještě jednou při obraceném využití tokenů. Druhý se musí rovnat a hledá se první tak, aby jeho editační vzdálenost byla menší než LIMIT. V případě že je nalezen vhodný bigram ze statistiky, je oprava automaticky přijata a nabídnuta uživateli. Pro následující token se tedy vhodná oprava nehledá, protože ji pokrývají již tento bigram. V opačném případě se pokračuje další částí algoritmu.

4.5.4. Oprava častých chyb

Tato část algoritmu se snaží podchytit některé jednoduché a časté chyby, kterých se uživatelé při zadávání dotazů dopouštějí. Jedná se většinou o pravopisné chyby nebo chyby způsobené prohozením některých znaků na klávesnici. Následující body popisují posloupnost, podle které algoritmus postupuje a hledá možné opravy.

1. Nedovolené znaky na začátku nebo konci řetězce (slovník)

   [*fotky¨, mp3-, tapety)*]

Prvním typem takových chyb je výskyt nepísmenných (nečíselných) znaků na začátku nebo častěji na konci řetězce. K tomu dochází při nechtěném stisku

některé z kláves v okolí klávesy Enter, pokud chce uživatel odeslat napsaný dotaz. V tomto případě se token předá funkci, která se pokusí nedovolené znaky najít a odstranit. Takto upravený řetězec se následně vyhledá ve slovníku, a pokud je nalezen, je automaticky vybrán jako oprava daného tokenu.

2. Záměna českých znaků za čísla (slovník)

   [*pr8ce, p5ekvapen9, slun94ko*]

   K této chybě dochází v případě, když se uživatel pokusí napsat slovo obsahující diakritiku, ale nemá správně nastavené české rozložení klávesnice. Algoritmus se v takovém případě pokusí nahradit všechny číselné znaky v řetězci odpovídajícími českými znaky, a pokud je výsledný řetězec nalezen ve slovníku, je automaticky shledán jako vhodná oprava pro opravovaný token.

3. Záměna Y - Z (slovník)

   [*ydarma, tapetz, seynamka*]

   Tato častá chyba vzniká v případě, pokud je špatně nastaveno rozložení klávesnice. Pokus o opravu spočívá v zájemném nahrazení všech výskytů těchto dvou znaku (Y a Z) v řetězci. Takto upravený token je následně vyhledán ve slovníku a v případě úspěchu přidán do seznamu vhodných kandidátů na opravu.

4. Záměna I - Y (slovník)

   [*mobili, obrázki, ubitování*]

   Špatné psaní měkkého a tvrdého „i“ je jedna z nejčastějších pravopisných chyb, které se v českém jazyce vyskytují. Algoritmus v tomto případě postupně prochází řetězec znak po znaku a nahrazuje tvrdé „y“ za měkké a naopak. Vždy je tedy provedena změna jen v jednom znaku a nedochází k záměně všech znaků najednou. Po každé záměně je takto upravený token opět vyhledán ve slovníku a v případě nalezení je přidán do seznamu kandidátů na opravu.

5. Záměna S - Z (slovník)

   [*obrásky, shodnotit, basar*]

   Stejně jako v předchozím případě je i tato chyba v českém prostředí poměrně častá a to hlavně pokud se jedná o záměnu na místě prvního znaku. Oprava je založena na stejném principu jako v minulém případě. Řetězec se postupně prochází a jednotlivé znaky „s“ a „z“ se navzájem zaměňují. Takto upravený token je porovnán se slovníkem a je-li nalezen, je přidán do seznamu vhodných kandidátů.

6. Nedovolené znaky na začátku nebo konci řetězce (statistika)

Stejný případ jako v kroku 1 s tím rozdílem, že upravený řetězec je vyhledán ve statistikách a přidán do seznamu možných kandidátů. V tomto případě není automatický shledán jako správná oprava, jelikož statistika unigramů může a také obsahuje i nesprávně psané dotazy, které se do ní dostaly vzhledem ke své vysoké četnosti výskytu v záznamech dotazů.

7. Záměna Y - Z (statistika)

Proces opravy odpovídá kroku 3, ovšem výsledný řetězec je v tomto případě vyhledán ve statistikách a přidán jako kandidát na opravu.

8. Překlep při stisku sousední klávesy (statistika)

[*onlone, zdarna, ptaha*]

Klasickým překlepem je chyba, která nastává při stisku klávesy sousedící se „správnou“ klávesou. K tomu často dochází při rychlém psaní, pokud uživatel dobře neovládá psaní „poslepu“. V tomto případě se předpokládá výskyt pouze jedné takovéto chyby, takže se špatně napsaný řetězec liší od správného tvaru pouze v jednom znaku. V prvním kroku tedy algoritmus vyhledá všechny unigramy ze statistiky, které se od vstupního řetězce liší právě v jednom znaku. Takto vybrané unigramy se následně s řetězcem porovnají a zkoumá se, zda se neliší ve stisku sousední klávesy. Funkce má k dispozici mapu sousedících znaků, která respektuje klasické rozložení klávesnice. Pokud je takový případ nalezen, je unigram zařazen na seznam vhodných kandidátů na opravu.

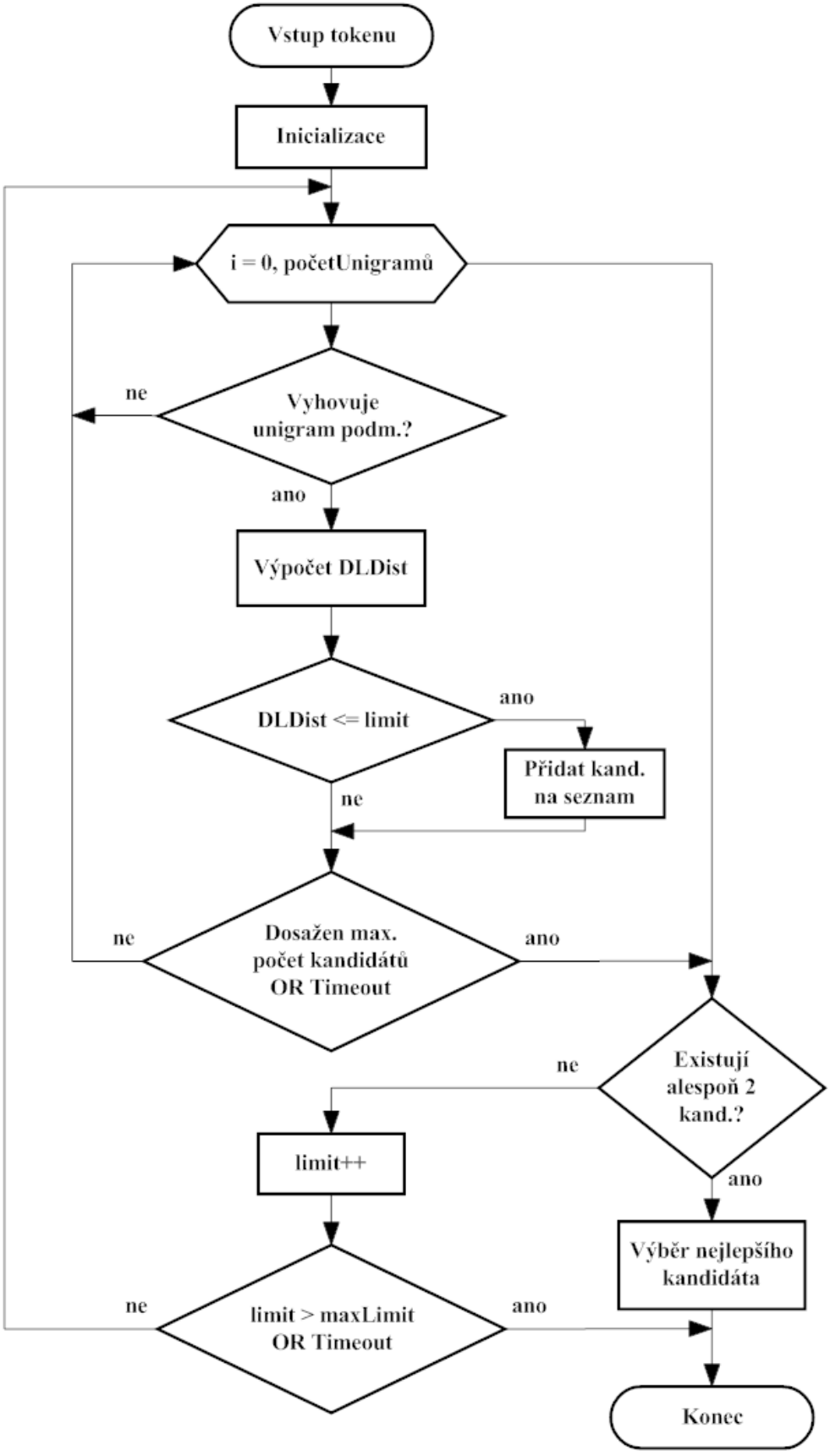
4.5.5. Kontrola seznamu kandidátů

Po provedení všech osmi kroků algoritmus poprvé zkontroluje seznam vhodných kandidátů na opravu, zda není prázdný. Pokud nalezne alespoň dva kandidáty, pokračuje algoritmus druhou částí a to výběrem nejvhodnějšího kandidáta. Možné návrhy na opravu musí být nejméně dva, a to proto, že jeden z nich je vždy samotné opravované slovo. V případě, že je v seznamu nalezen jen jeden kandidát, pokračuje proces vyhledávání další částí.

4.5.6. Damerau-Levenshteinova vzdálenost

Zatímco předchozí část algoritmu sloužila pro zachycení některých konkrétních případů, které vedou ke vzniku překlepu, snaží se tato metoda vyhledat ve statistikách unigramů všechna slova, která vykazují určitou podobnost s opravovaným. Tato podobnost je vyjádřena editační vzdáleností mezi oběma slovy a určuje, kolik elementárních editačních operací je potřeba k přetvoření jednoho slova v druhé. Čím nižší je tato vzdálenost, tím existuje větší pravděpodobnost, že se jedná o správnou opravu. V této části je také nutné neustále kontrolovat dobu provádění algoritmu, protože ta nesmí přesáhnout maximální povolenou dobu pro hledání opravy. Pokud není nalezen žádný vyhovující kandidát během vymezeného času, není pro dotaz navržena žádna oprava. Tento případ může nastat, pokud dotaz obsahuje mnoho překlepů nebo je velmi dlouhý. Průběh hledání kandidátů pomocí Damerau-Levenshteinovy vzdálenosti je následující:

1. nastaví se maximální hodnota pro editační vzdálenost (MAX_LIMIT), aktuální LIMIT se nastaví na 1
2. for I = 0 to POCET_UNIGRAMU - postupně se prochází pole unigramů seřazené sestupně podle četnosti
3. zkontroluje se, zda četnost aktuálního unigramu je X krát větší než vstupního (opravovaného), pokud ne pokračuje se krokem 8
4. pokud se délka vstupního a aktuálního unigramu liší o více než LIMIT znaků, pokračuje se dalším unigramem
5. vypočítá se Damerau-Levenshteinovy vzdálenost mezi unigramy, pokud je menší než LIMIT, přidá se aktuální unigram na seznam možných kandidátů
6. zkontroluje se počet kandidátů na seznamu, je-li nižší než dovolený maximální počet kandidátů, pokračuje se dalším unigramem; v opačném případě se jde na krok 8
7. zkontroluje se doba vykonávání algoritmu, pokud nebyl překročen časový limit, pokračuje se dalším unigramem; jinak krok 8
8. pokud seznam kandidátů obsahuje alespoň dva návrhy, přejde algoritmus do části výběru nejlepšího z nich → konec;
9. inkrementuje se LIMIT, pokud není větší než MAX_LIMIT, pokračuje se krokem 3 a celé pole se znovu projde; jinak není oprava nalezena → konec



Obr. 4.5: Průběh návrhu kandidátů pomocí Damerau-Levenshteinova vzdálenost (DLDist)

4.5.7. Výsledek vyhledávání

Po provedení všech pokusů o nalezení vhodných kandidátů na opravu vstupního tokenu může nastat jedna z následujících variant:

1. Nebyl nalezen žádný jiný kandidát na opravu než samotné opravované slovo ať už z důvodu, že neexistuje žádné podobné slovo, nebo vypršel časový limit pro vykonání opravy. V takovém případě se zachová token v původním tvaru.
2. Byla nalezena oprava, kterou je možné jednoznačně schválit jako správnou (například po odstranění nedovolených znaků). Automaticky tak dojde k nahrazení vstupního tokenu touto opravou.
3. Seznam možných oprav obsahuje více než jednoho kandidáta. To znamená, že existuje alespoň jedno slovo, které by mohlo představovat korektní tvar vstupního tokenu, pokud obsahuje překlep. Takových kandidátů může být i více, takže je potřeba rozhodnout, který z nich je nejvhodnější, případně zda je vůbec potřeba vstupní token opravovat. Tento problém řeší druhá část algoritmu, jejíž popis následuje.

4.6. Výběr nejlepšího kandidáta

Pokud algoritmus dospěje až do této části, je zřejmé, že seznam kandidátů obsahuje alespoň dvě položky, ze kterých je potřeba určit tu nejvhodnější. Tento úkol je velmi komplikovaný a to hlavně z důvodu, že se algoritmus snaží opravovat i neslovníkové výrazy (dotazy), a proto neexistuje stoprocentní jistota, zda je nutné vstupní slovo vůbec opravovat, nebo zda navržená oprava skutečně vystihuje původní záměr uživatele, který se i při zadávání dotazu překlepl.

4.6.1. Systém hodnocení kandidátů

Pro výběr nejlepší opravy jsem zavedl určitý systém hodnocení, který každého kandidáta oboduje podle stanovených kritérií a následně vybere toho s nejvyšším počtem bodů. Takto vybraný kandidát může reprezentovat nejvhodnější opravu, nebo samotné opravované slovo, které se na seznamu také nachází. V takovém případě není nabídnuta žádná korekce a token se zachová.

Seznam kritérií, která se vyhodnocují, a jejich bodové ohodnocení představuje tabulka 4.1. Důležité je správné rozložení bodů za splnění jednotlivých

měřítek, protože každé má z hlediska výběru nejlepší opravy jinou váhu. Čím vyšší je bodové ohodnocení, tím má hodnocené kritérium větší význam.

Konkrétní množství bodů udělených za splnění jednotlivých kritérií jsem určil experimentální metodou. Vždy jsem měnil jednotlivé hodnoty a následně zjišťoval vliv na přesnost opravy, kterou algoritmus poskytoval. K testování jsem využil soubor dotazů, který obsahoval jak správné dotazy bez překlepu, tak ty chybné, které bylo nutno opravit (více v části 6).

kritérium	**ohodnocení [b]**
přítomnost v CZ slovníku	30
přítomnost v EN slovníku	20
délka kandidáta > délka oprav. tokenu	80
první znak kandidáta = první znak oprav. tokenu	100
četnost kandidáta	$\frac{\text{četnost kandidáta} * 150}{\text{max. četnost}}$
přítomnost v CZ slovníku (pouze vstupní token)	*200*
přítomnost v EN slovníku (pouze vstupní token)	*100*

Tab. 4.1: Přehled hodnocených kriterií

Kritérium, které hodnotí délku kandidáta, se snaží zvýhodnit překlepy vzniklé přidáním znaku do slova. Algoritmus se již v první fázi návrhu kandidátů snaží zachytit, překlepy vznikající stiskem sousední klávesy. Takových případů se vyskytuje ve skutečnosti velké množství. Naopak k záměně znaku, který se na klávesnici nevyskytuje v sousedství té správné, dochází jen zřídka.

Bodové ohodnocení za četnost, s jakou se kandidát na opravu vyskytuje v záznamech dotazů, se pohybuje v rozmezí 0 až 150 bodů. Ten s nejvyšší četností získá 150 bodů a všichni ostatní pak přímo úměrně své četnosti. Toto hodnocení zvýhodňuje výrazy, které se často vyskytují v dotazech zadávaných do vyhledávače a tedy je větší pravděpodobnost, že se jedná o korektní opravu.

Poslední dva řádky tabulky udávají počet bodů, které by získal opravovaný token, pokud by nebyl automaticky shledán jako správný po nalezení ve slovníku, ale byl by zařazen jako kandidát do seznamu. Bodové ohodnocení je vysoké, protože je velká pravděpodobnost, že se jedná o korektní výraz a oprava není potřeba.

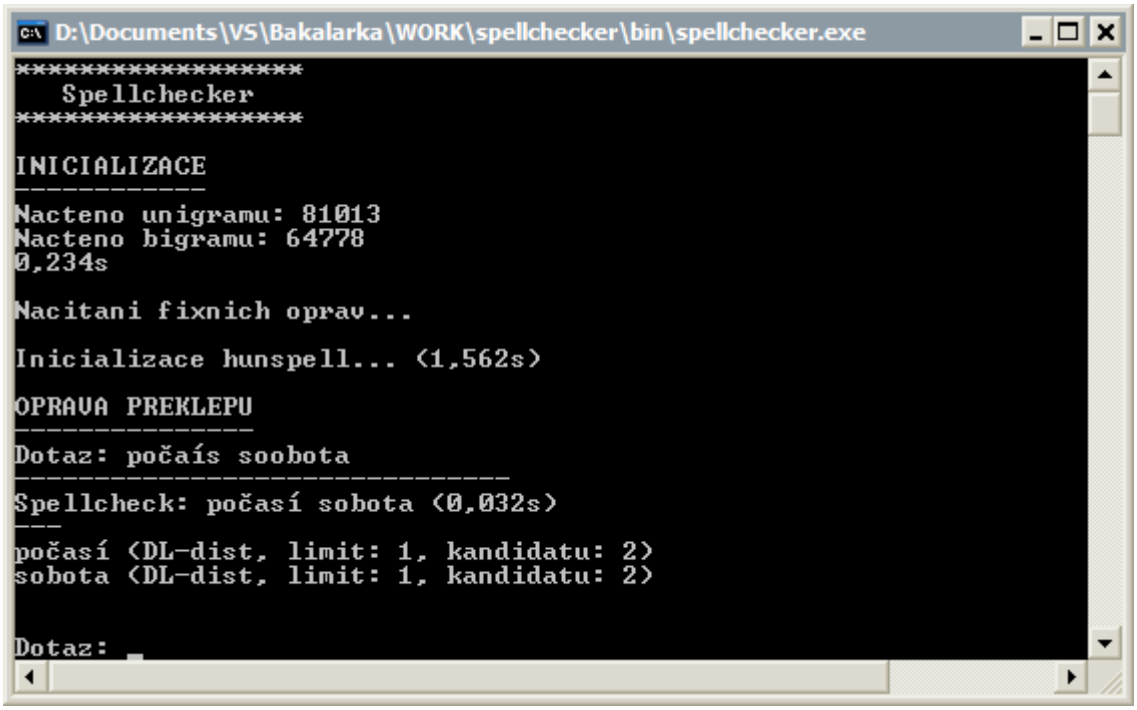
5. Implementace algoritmu

5.1. Použité prostředky

Důležitou součástí této práce je také funkční implementace navrženého algoritmu a to v některém programovacím jazyce. Vytvoření tohoto prototypu je nezbytné zejména z důvodu vyhodnocení úspěšnosti algoritmu, z čehož vychází i možnost dalšího zdokonalování a vylepšování. Je také možné sledovat reálnou rychlost algoritmu, která je v tomto případě velmi podstatná.

Vzhledem k nárokům na rychlost není možné provést implementaci v jakémkoliv interpretovaném jazyce jako je PHP, Perl, Python a podobně, jelikož je vykonávání skriptů napsaných v takovém jazyce velmi pomalé. Je nutné použít kompilovaný nízkoúrovňový jazyk, který má co nejblíže vlastnímu strojovému jazyku počítače a který je možné efektivním způsobem propojit s webserverem, jenž bude poskytovat dotazy zadávané do vyhledávače. Těmto požadavkům nejlépe vyhovuje jazyk C, a proto jsem ho použil pro vlastní implementaci algoritmu. Jako vývojové prostředí jsem s úspěchem využil Visual Studio 2008 od firmy Microsoft.

Aplikace včetně zdrojových kódů je umístěna na přiloženém CD.



Obr. 5.1: Náhled prototypové aplikace

5.2. Prototypová aplikace

5.2.1. Popis aplikace

Samotný prototyp, na kterém by bylo možné vyzkoušet a otestovat úspěšnost algoritmu, představovala konzolová aplikace spustitelná pod systémem Windows. Program očekává zadání dotazu od uživatele, který má simulovat dotaz zadaný do internetového vyhledávače. Ten je následně předán algoritmu, vyhodnocen a výstup je zobrazen na obrazovce. Výstup obsahuje kromě samotné navržené opravy také některé doplňující informace o samotném průběhu opravy. Mezi ně patří pravidlo, podle kterého k opravě došlo, celkový počet kandidátů na opravu a také čas potřebný k opravě dotazu. Volitelně je také možné zapnout zobrazování hodnocení jednotlivých kandidátů. Aplikace po zpracování dotazu očekávání zadání dalšího a ukončí se až po zadání prázdného řetězce.

Součástí aplikace je také modul, který lze použít pro zpracování surových záznamu dotazů (soubory je nutné umístit do složky *logs*). Ten nejdříve rozdělí dotazy na jednotlivé unigramy a bigramy, následně spočítá jejich četnost a výsledek je uložen do souboru. Tímto způsobem je možné rozšířit statistický soubor, ze kterého algoritmus při opravě překlepů vychází.

5.2.2. Zdrojová data

Samotná aplikace je tvořena pouze čistou implementací navrženého algoritmu, a proto je nutné někde uchovávat data, ze kterých algoritmus při opravě vychází. K tomu slouží několik složek, které tyto zdroje obsahují, a jsou proto nezbytné pro běh aplikace. Složky jsou dvě a nacházejí se přímo v hlavní složce s programem.

První má název *dict* a obsahuje zdrojová data pro slovník Hunspell. Celkem se zde nachází čtyři soubory a to pro každý jazyk dva (čeština a angličtina). Soubory s příponou *dic* (*hunspell_cs_CZ.dic*) obsahují všechna slova v základním tvaru pro daný jazyk a soubory s příponou *aff* (*hunspell_cs_CZ.aff*) pak pravidla, podle kterých se tato slova ohýbají do dalších povolených tvarů.

Druhá nezbytná složka má název *stats* a jejím obsahem jsou statistiky unigramů (*unigram_stats.txt*) a bigramů (*bigram_stats.txt*), které vznikly zpracováním záznamů dotazů. V této složce se nachází také zdrojový soubor pro stálé opravy (*fixed_corrections.txt*).

6. Vyhodnocení úspěšnosti algoritmu

Vyhodnocení algoritmu z hlediska úspěšnosti je velmi důležitá součást celého návrhu. Slouží především jako vodítko, které umožňuje algoritmus dále upravovat a vylepšovat. Je tak možné sledovat, jak úspěšnost algoritmu závisí na jeho jednotlivých částech, například jakou výhodu přináší použití slovníku nebo využití bigramů při návrhu možných kandidátů.

Vysoká úspěšnost je také nutná pro reálné nasazení ve vlastním vyhledávači. Není možné uživateli nabízet nesmyslné návrhy na opravu v případě, že se o překlep nejedná, nebo naopak nenabízet žádnou opravu v případě jasných chyb. Určitá míra nepřesnosti se ale u algoritmu, který nevyužívá pouze stále opravy, bude vždy vyskytovat. Snaha je tuto chybu co nejvíce minimalizovat, aby algoritmus přinášel uspokojivé výsledky.

6.1. Způsob vyhodnocení

Úspěšnost algoritmu jsem testoval pomocí klasické metody pro zjištění přesnosti (accuracy). Ta se v tomto případě rovná poměru správně opravených dotazů ku celkovému počtu dotazů. Mezi správně opravené dotazy lze počítat případy, kdy byl vstupní dotaz skutečně chybný a algoritmu nabídnul správnou opravu a případy, kdy algoritmus rozpoznal, že oprava dotazu není potřeba.

Přesnost algoritmu je možné hodnotit ze dvou hledisek. První je úspěšnost opravy skutečně chybných dotazů, kdy algoritmus vybere ze všech možných kandidátů správnou korekci. Druhým hlediskem je pak úspěšnost při rozpoznávání, zda je vůbec nutné zadaný dotaz nějakým způsobem opravovat. Tomu také odpovídají data, na kterých jsem algoritmus testoval.

6.2. Testovací data

Úspěšnost algoritmu jsem testoval na dvou souborech dat, které jsem získal ve spolupráci se Seznamem.cz. Jedná se o skutečné dotazy, které uživatelé zadávají do vyhledávače, a proto zjištěná úspěšnost co nejvíce odpovídá reálnému nasazení, i když se jedná pouze o zlomek objemu, který musí vyhledávač každodenně zpracovat.

První testovací soubor (*TS 1*) se skládal celkem z 1190 dotazů, z čehož bylo 1120 správných a 70 chybných, které potřebují opravit. Tento vzorek přibližně vystihuje poměr, ve kterém uživatelé zadávají do vyhledávače správné a chybné dotaz, a proto je na něm možné testovat celkovou úspěšnost algoritmu. Druhý soubor (*TS 2*) pak obsahoval pouze 247 dotazů, ale všechny byly chybné.

Pro přesnější vyhodnocení by byl samozřejmě potřeba větší objem testovacích dat. Jejich získání je ovšem obtížné, protože je potřeba je sestavit ručně a to zabere nemalý čas. Pro představu o kvalitách algoritmu však použitý vzorek stačí.

6.3. Výsledky vyhodnocení

Vyhodnocení algoritmu je zachyceno v následujících dvou tabulkách. Tabulka 6.1 ukazuje celkovou úspěšnost navrženého algoritmu. Celková přesnost opravy (soubor TS 1) je zde 90,67 %, což představuje velmi dobrý výsledek.

		TS 1	**TS 2**
celkem dotazů	*G*	1120	0
	B	70	247
opraveno správně	*GR*	1045	0
	GC	34	199
opraveno špatně	*BR*	75	0
	BC	7	14
	NC	29	34
přesnost opravy chyb		**48,57 %**	**80,57 %**
celková přesnost		**90,67 %**	**80,57 %**

G počet korektních dotazů, které nepotřebují opravu
B počet chybných dotazů
GR správně rozeznané korektní dotazy
GC správně opravené dotazy obsahují chybu
BR špatně rozeznané korektní dotazy
BC špatně opravené dotazy obsahující chybu
NC neopravené dotazy obsahující chybu

Tab. 6.1: Přesnost finální verze algoritmu

Následující tabulka pak zobrazuje úspěšnost algoritmu, pokud se zakáže využití důvěryhodného slovníku. V tomto případě dochází k velkému nárůstu počtu špatně rozeznaných dotazů, které neobsahují chybu, a není je nutné jakkoliv opravovat. Celková přesnost algoritmu tím klesla na 76,64 %, což je již spíše podprůměrná hodnota, a proto je využití slovníku nezbytné.

		TS 1	**TS 2**
celkem dotazů	*G*	1120	0
	B	70	247
opraveno správně	*GR*	882	0
	GC	30	206
opraveno špatně	*BR*	238	0
	BC	18	19
	NC	22	22
přesnost opravy chyb		**42,86 %**	**83,40 %**
celková přesnost		**76,64 %**	**83,40 %**

Tab. 6.2: Přesnost algoritmu bez využití slovníku

6.4. Možnosti zlepšení

Celkově se úspěšnost algoritmu jeví jako dostatečná, existuje však několik oblastí, kde by bylo potřeba ještě zapracovat. Po podrobném prozkoumaní výsledků, kdy došlo k nesprávné opravě, jsem nalezl několik problémů.

Prvním z nich je špatné využití bigramů, kdy často dochází k chybné opravě jinak korektního výrazu. V tomto případě by bylo nutné zapracovat více podmínek, kdy se může oprava pomocí bigramů aplikovat a také lépe analyzovat pravděpodobnost, zda je možné je v daném případě použít. V úvahu také připadá využití trigramů, které by bylo nutné nejdříve získat ze záznamů dotazů.

V některých případech není ideální výběr nejlepšího kandidáta. Proto by bylo žádoucí zapracovat některá další kritéria, která se při výběru uplatní. Například minimální bodový odstup nejlepšího kandidáta od ostatních, aby byl uznán jako možná oprava.

Algoritmu by měl také řešit chybnou segmentaci dotazu, kdy uživatel některá slova spojí dohromady, nebo naopak rozdělí výrazy, které by měly tvořit jedno slovo. Tuto funkci by nemělo být obtížné implementovat.

7. Závěr

Cílem této práce bylo navrhnout algoritmus pro opravu překlepů dotazů zadávaných do vyhledávače. Zadavatel byla firma Seznam.cz, která provozuje jeden z největších internetových vyhledávačů v České republice a která také poskytla potřebná data a spolupracovala při samotné realizaci.

Motivací pro návrh nového algoritmu byla skutečnost, že tradiční opravu překlepů nelze pro webové dotazy s úspěchem použít. Tyto dotazy se totiž v několika aspektech liší od běžně psaného textu a jiné jsou také časové nároky na algoritmus. Celkově neexistuje mnoho vyhledávačů, které mají integrovanou opravu překlepů, a pokud ano, jsou technické podrobnosti neveřejné a nedostupné, jelikož jim to poskytuje určitý náskok před ostatními vyhledávači.

Hlavní podmínkou, kterou musel navržený algoritmus splnit, byla schopnost opravovat i neslovníkové výrazy. Proto bylo nejdříve nutné zpracovat Seznamem poskytnuté záznamy dotazů, které uživatelé do vyhledávače zadávají. Jelikož se jednalo o velké množství dat, vznikl tak první úkol, který bylo nutné vyřešit. Výsledkem byl soubor dat, který tvoří základ algoritmu.

Vlastní návrh algoritmu nevychází z žádného již použitého řešení, ale je kompletně vystavěný od nuly. Základ tvoří důvěryhodný slovník, který jasně vymezuje neznámá slova, jež se následně podrobí procesu opravy. V první fázi se podle několika pravidel a s využitím Damerau-Levenshteinovy vzdálenosti hledají vhodní kandidáti na opravu. Druhá fáze má poté za úkol vybrat nejvhodnější opravu případně zachovat původní slovo, pokud žádná nevyhovuje. Pro účely vyhodnocení byl následně navržený algoritmus implementován v podobě konzolové aplikace, která realizuje opravu vstupních dotazů.

Podařilo se tedy splnit hlavní cíl práce a to navrhnout a implementovat algoritmus pro opravu překlepů. Následné vyhodnocení pak zhodnotilo kvality algoritmu a jeho případné šance pro reálné nasazení ve vyhledávači. V hlavním testu přesáhla úspěšnost opravy hranici 90 %, což lze hodnotit jako velmi dobrý výsledek. Také schopnost opravit kolem 80 % chybných dotazů je dostačující. Algoritmus byl testován na velmi malém vzorku dat, která ale odpovídají složení internetových dotazů, takže je výsledek celkem vypovídající.

Skutečnému nasazení má ale algoritmus celkem daleko, jelikož je potřeba ještě vylepšit několik věcí. Mezi ty zásadní patří lepší využití bigramů a zpřesnění výběru nejlepšího kandidáta. Také je nutné rozšířit soubor dat, ze kterých algoritmus při opravě vychází, protože pro potřeby práce byly zpracovány záznamy pouze za období 14 dnů. Výrazné zlepšení úspěšnosti algoritmu je možné pomocí rozšíření množství stálých oprav, které by pokrylo ty nejčastější chyby, které se v dotazech vyskytují. K tomu by bylo potřeba využít zpětnou vazbu a sledovat, jaké opravy uživatelé nejčastěji využívají a podle toho seznam stálých oprav rozšiřovat.

Řešení navržené v této práci může sloužit jako dobrý základ pro skutečnou implementaci opravy překlepů do vyhledávače. Bohužel firma Seznam.cz si již v průběhu práce vymyslela vlastní řešení, které následně i implementovala. Navržený algoritmus může tedy posloužit k případnému porovnání úspěšnosti a dalšímu vylepšení současného systému, nebo jako základ pro opravu překlepů v jiném vyhledávači.

Seznam použití literatury

[1] Bentley, Jon – Sedgewick, Bob. *Ternary Search Trees* [online]. Dr. Dobbs Journal, 1. dubna 1998. URL:‹www.ddj.com/windows/184410528›.

[2] Pfaff, Ben. *Binary Search Trees* [online]. GNU libavl 2.0.2. URL:‹www.stanford.edu/~blp/avl/libavl.html/Binary-Search-Trees.html›.

[3] Cucerzan, Silviu – Brill, Eric. *Spelling correction as an iterative process that exploits the collective knowledge of web users* [online]. Microsoft Research, Microsoft One Way. URL:‹http://research.microsoft.com/users/silviu/Papers/emnlp04.pdf›.

[4] Hal Daum'e III – Brill, Eric. *Web Search Intent Induction via Automatic Query Reformulation* [online]. Microsoft Research, Microsoft One Way. URL:‹http://research.microsoft.com/tmsn/Papers/HLT_USER_INTENTS.pdf›.

[5] Martins, Bruno – Silva, Mário J. *Spelling Correction for Search Engine Queries* [online]. Departamento de Informática, Faculdade de Ciências da Universidade de Lisboa. URL:‹ http://xldb.fc.ul.pt/data/Publications_attach/spellcheck.pdf›.

[6] *Levenshtein distance* [online]. Wikipedia.org, poslední aktualizace 5. 5. 2008. URL:‹http://en.wikipedia.org/wiki/Levenshtein_distance›.

[7] *Algorithm implementation/Strings/Levenshtein distance* [online]. Wikibooks, poslední aktualizace 22. 4. 2008. URL:‹http://en.wikibooks.org/wiki/Algorithm_ implementation/Strings/Levenshtein_distance›.

[8] *N-gram* [online]. Wikipedia.org, poslední aktualizace 24. 4. 2008. URL:‹http://en.wikipedia.org/wiki/N-gram›.

[9] *Hamming distance* [online]. Wikipedia.org, poslední aktualizace 5. 5. 2008. URL:‹http://en.wikipedia.org/wiki/Hamming_distance›.

[10] *Soundex* [online]. Wikipedia.org, poslední aktualizace 14. 3. 2008. URL:‹http://en.wikipedia.org/wiki/Soundex›.

[11] Németh, László. *Hunspell* [online]. URL:‹http://hunspell.sourceforge.net/›.